Boot Command Schema

And a mocked Process Manager

Brett Viren

March 16, 2021

Goals

- Develop and initial schema for a `boot` command.
  - Exercise it with a mock up of a Process Manager that matches my understanding of Nomad.
- Collect feedback for revision.

The boot command

The CCM boot *command provides* ***high-level*** *description of the* ***desired set of processes*** *in a* ***DAQ partition*** *of a* ***particular type****.*

Notes

- A boot may be "realized" by starting a number of processes or *tasks*.
- Use of **type** means boot describes the general makeup of a partition.

The boot does **not** provide:

- specific process construction/identity (that is in init)
- specific process/module configuration (that is in conf)
- a run number (that is in start)

Expectations/assumptions of use of boot

$$\mathrm{RC}(\{cmds\}) \rightarrow [\mathrm{boot}] \rightarrow \mathrm{PM}(\mathrm{env}) \rightarrow \{tasks\}$$
$$\mathrm{RC} \leftarrow (\text{resolution info}) \leftarrow \mathrm{PM}$$
$$\mathrm{RC} \rightarrow [\mathrm{init}] \rightarrow \{tasks\}$$
$$\mathrm{RC} \rightarrow [\mathrm{conf}] \rightarrow \{tasks\}$$

- Human operators write boot and other command objects.
  - initially by-hand/script, eventually with help of web UI
- RC will deliver boot to the Process Manager[1] (PM)
- PM **realizes** the partition by an "inner join" of boot info with **DAQ environment** info.
  - *eg*, available computers and other resources, user accounts
- PM must **provide back** information it **resolves** in the process of **realizing** the boot.
  - In part, info to allow RC to find REST CF's of the $\{tasks\}$

[1] Assume PM is something like Nomad.

Top `boot` schema

So far, `boot` object has two attributes:

`ident` A unique identifier for the **type** of partition.

- For now a free `string`. Could encode aggregate values, or be an explicit `record`. Could express version information.

`jobs` A sequence of `job` objects which describe the desired partition.

What is a CCM `job` object?

*A CCM `job` object provides **high-level** description of some **desired state** realized as a related **set of tasks** (processes).*

- Takes Nomad's definition of "job".
- Is defined in abstracted terms.
  - no specific user or host names / ports
- PM will **realize** a `job` object as a set of running processes (*tasks*).
  - in the process, **resolve** specific user/host/port/etc.

The job schema

A job object so far has these top attributes:

- ident Uniquely identify the job in the partition (ie, in boot).
- roles A sequence of identifiers from a known set.

cardinality The number of tasks to realize for this job.

parameters Sequence of key/value role qualifiers.

The `job.roles`

*A CCM `role` names an **aspect** of the `job`'s tasks. A `role` may have zero or more `parameters`.*

- We must develop a `role` taxonomy as we develop the DAQ.
- A `job`'s set of `roles` is interpreted by PM to realize the `job`'s *tasks*.
- Each `role` a `job` asserts implies:
  - a set of `parameters` which may be included in the `job` to provide **qualifying information**.
    - a `role` is effectively a functional transformation on the `job` and the parameters are functional arguments
  - a **data structure** (following a schema) provided by PM back to RC giving any and all info **resolved** from the abstract `role` and `parameters`
    - again, *eg*, giving host/port of REST CF

Example `role` values

- `"appfwk"` translates to a `daq_application` command line and implies PM must return the `hostname` and `port` for the REST Command Facility of each task.
- `"zoned"` translates to requesting some class of resource and may influence how PM selects `hostname` and `port` and how/where it launches the processes. The `job` may provide a `parameter`: `zone="local"` or `zone="upstream"`, `affinity="APA42"`.

Just examples, I'm sure a lot of bikeshedding and more serious invention will be required.

The boot schema

Current try at a boot schema as described is a moo example:

`moo/examples/still/still-boot-schema.jsonnet`

Reflects what was just described.

Goals of the “still” mock up

- Not intended for actual use outside initial schema vetting
- Tries to mimic how I think input to Nomad will work
  - I may have it wrong, expect to iterate
- Wanted something concrete to sanity check the schema
- The mock’s details are not so important but are in backup slides

```
$ cd moo

# make boot object for "partition type 42"
$ python examples/still/mkboot.py

# make PM->RC return object and tasks' command lines
$ python examples/still/fakepm.py p42-boot.json

$ cat p42-data.json       # "run control" goes here

# realize the partition
$ shoreman Procfile.p42   #  "process management"
```

Summary and next steps

- A description of `boot` is given. Agreement?
- An initial matching schema is available.
- A mock up for how Nomad will consume `boot` exists.
- If there are no show stoppers we next:
  - implement `boot` for actual partition types
  - "impedance match" schema with Nomad expectations
  - implement initial Nomad consumption and realization
  - over time, extend schema with new `roles` and `parameters`
  - add `boot` object creation to Web UI scope

Mock produce a `boot`

A mock of eventual scripts and web UI:

```
$ python examples/still/mkboot.py
wrote: p42-boot.json
```

- Produced file holds a `boot` object.
- The mock includes 2 `job` objects with cardinality > 1
- One job simply specifies a role interpreted simply run `sleep` commands.
- One run specifies `roles=["appfwk", "zoned"]` with `"zone"` parameter `"remote"`.

Mock consume a `boot` - realize partition

A mock of my partition realization by creating a foreman/shoreman Procfile:

```
$ python examples/still/fakepm.py p42-boot.json
wrote: Procfile.p42
wrote: p42-data.json
$ shoreman Procfile.p42
```

The `shoreman` script simply runs the listed tasks, maybe SSH'ing for any `zone="remote"`.

Again, I try to mimic how I think Nomad works, but with less bells and whistles.

Mock consume a `boot` - resolved info

Mock Nomad by creating a JSON file holding the "resolved" information.

```
$ jq '.[0][0].params' < p42-data.json
{
  "zone": "local",
  "sleeps": "20",
  "user": "bv",
  "port": 9001,
  "hostname": "localhost"
}
```

RC would then use this to, *eg*, learn REST Command Facility URL.